PERSONAL PASCAL

Compiled PASCAL for the IBM Personal Computer®

DAVID E. CORTESI

GEORGE W. CHERRY

Personal Pascal

Compiled Pascal for the IBM Personal Computer®

David E. Cortesi

George W. Cherry

A Reston Computer Group Book
Reston Publishing Co., Inc.
A Prentice-Hall Company
Reston, Virginia

Library of Congress Cataloging in Publication Data
Cortesi, David E.
Personal Pascal.

Includes index.
1. IBM Personal Computer—Programming. 2. PASCAL (Computer program language) I. Cherry, George William. II. Title
QA76.8.I2594C66 1983 001.64'2 83-8686
ISBN 0-8359-5523-0
ISBN 0-8359-5522-2 (pbk.)

A Prentice-Hall Company
Reston, Virginia 22090

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Preface

This book is a text on the use of Pascal in general, and compiled Pascal on the IBM Personal Computer in particular. Its aim is to teach effective use of Pascal as a compiled programming language, when that use will take place in an environment composed of an IBM Personal Computer, using either the IBM Pascal compiler or the Digital Research Pascal/MT+86 compiler.

Beginning programmers encounter several kinds of difficulties:

- Getting started
- Finishing
- Avoiding syntactical errors

A helpful textbook assists its reader in these areas. This book helps the reader to get started by demonstrating the top-down strategy of design. It helps the reader complete a design by illustrating stepwise refinement. And it helps the reader avoid common syntactic errors by giving careful descriptions and illustrations of Pascal syntax. Clarity of design and portability of code are emphasized throughout.

The first two chapters introduce the novice programmer to simple program statements and to simple, complete programs. The third chapter explains in detail how the compilers are used to translate programs. From that point forward, the reader should be able to compile and run each example program in the text. The reader who already knows another programming language and wants to pick up Pascal can skip chapter 1 entirely and skim chapter 2. Those who know their Personal Computer's operating system well will not have to linger in chapter 3.

Chapters 4–13 cover the *entire* (standard) Pascal language, including textfiles (chapter 5), records (chapter 10), pointers and dynamic variables (chapter 11), files (chapter 12), and sets (chapter 13). The central themes in these chapters are, first, to provide a careful, thorough explication of

the syntax of Pascal and, second, to demonstrate the use of the language through many example programs and program fragments. The language's syntax is also summarized for reference in Appendix A, using a modified (and more readable) form of BNF recommended by Ledgard.

The last two chapters are devoted to describing the specific quirks, advantages, and nonstandard features of the two compilers. Appendix B summarizes the use of the ASCII code with the IBM Monochrome Display and the IBM printer. Appendix C summarizes standard input/output; Appendix D, standard procedures; and Appendix E, the IBM system of character widths.

This book is based, in large part, on *Pascal Programming Structures* by George Cherry (Reston, 1981). It is to George Cherry that the book owes its depth and its pedagogical virtues; any defects in the present version are entirely my own. I am grateful to Cherry for laying such a solid foundation for the work, and to Reston Publishing Company for giving me the opportunity to officiate at this wedding of an outstanding text with a computer of equal quality.

Contents

1 Introduction to Programming

OBJECTS AND ACTIONS

A computer program is a sequence of instructions to a computer processor, telling it to perform useful *actions* on significant *objects*. Early in the history of computers the objects were almost always numbers, and the actions on them were almost always the operations of arithmetic. This is no longer the case. For example, there are computer programs that help us to create, edit, and typeset English text. In such programs the objects are letters, punctuation marks, words, and paragraphs; the useful actions include insertion, deletion, pagination, and printing.

The significant objects can also be lines, curves, and other elements of figures; examples of the useful actions on such objects include position, rotate, change color, move, erase, and print. Objects and actions like these are used in computer graphics and in programs for computer-aided drafting and design.

Another example of significant objects is that of records that are arranged in lists—an insurance company's list of records of its policyholders, for instance. The useful actions here are manipulations on the data in the records—to print the names of all policyholders who had more than three claims in the last five years; to print reminder notices to all policyholders whose premiums are thirty days overdue; or to send information about automobile insurance to all the policyholders who have life insurance and no automobile insurance.

DATA

We call the significant objects on which a program is to work, *data*. The first ingredient of a well-written program is a thorough and careful

description of the types of data on which the program is to act. One important advantage of the Pascal programming language is that it lets us describe the *objects* of a program as clearly as the *actions* to be done on them.

Constants

There are two kinds of data in a computer program. Data whose values cannot change during the execution of the program are called *constants*. For example, the number of ounces in a quart, 32, would be a constant in any reasonable program.

Variables

Data whose values can change are called *variables*. Variables are the focus of most of a program's actions. At the insurance company, a program might well contain a variable named `PolicyHolder`. The value of `PolicyHolder` will probably not be constant. Rather, it will contain information about the one particular policyholder the program is acting on at any particular moment. When the program moves on to consider a different policyholder, it will arrange for information about that different person to be associated with the name `PolicyHolder`.

The Type of Data

Every variable in a Pascal program has a distinct and unchanging data *type* associated with it. A data type is a statement of what *kind* of data a variable may contain. It determines what values the variable can reasonably hold, what operations may validly be performed on the data, and the form and arrangement of the data in computer storage. Definitions of data types are a prominent feature of most Pascal programs. They give the program's author a tool for controlling the program's shape, and they give the program's reader a firm grasp on the author's intent.

INSTRUCTING A COMPUTER

Since computers don't "think" the way we do (or at all), there is always some compromise when we human beings try to communicate with a computer. We would like to instruct the computer the way we would talk to a human assistant, saying things like "Keep a record of the hours, overtime, and bonuses of each employee and print out their paychecks each month," or "Keep an inventory of all the new automobiles we have in stock and tell me right now how many air-conditioned, manual transmission, two-door hatchbacks we have," or "Keep a record of the sales by all salespeople and print out their names and sales-to-date in descending order." Unfortunately, we can't do that.

The Communication Gap

Despite the movies in which actors carry on colloquial dialogues with their computers, you probably know that there's a vast gulf between what we would like to say to the computer and what the computer can understand. The computer, after all, works only with strings of binary digits (or "bits") like 11010001, 00100101, 11100111, and so on. It has no means of working with the consequential objects like `PolicyHolder` or `PageNumber` that we use. The computer's native actions are primitive and elementary, unrelated in any way we can readily see to the problem-solving actions that we want done. If we didn't already know that computers are useful, we might despair that they ever could be.

Programming Languages

We bridge this formidable gap with a *programming language*. A computer programming language is a notation for describing the objects we want to work with, and the actions that we want the computer processor to perform on them. A programming language is a simple, rigidly defined, formal notation that is easy to translate into the machine's language of binary numbers. Nevertheless, a good programming language makes it easy for us to express our problem solutions; its notational forms are well suited to the description of the kinds of objects and actions that we want the computer to deal with.

In order to instruct a computer, we write a program in a programming language, and store that program as a file in the machine's file medium—on a diskette, in the IBM Personal Computer. Then we run a computer program called a *compiler,* which performs the translation of our program into the computer's internal language. The translated program—a sequence of binary numbers—is also stored in a file, and it can then be loaded into the machine and executed. We use the computer to translate a language that we can use into one that it can use. Thus does the computer help solve the communication problem which it created!

The process of translating from a programming language into a machine language is called *compiling* a program. It is a complex process, even though programming languages are more simple and limited than natural languages like English. A good compiler for the Pascal programming language will do more than translate your program for you. It will also check your program's adherence to Pascal's rules for data typing and syntax, and make whatever checks it can of the program's logical consistency. A good compiler is the programmer's best friend. The Pascal compilers available for the IBM Personal Computer are both of good quality.

Computers and programs may seem to be at the outer edge of high technology, but the ideas behind them are common and homely. The recipe for a cake, the instructions for knitting a sweater, or the directions for building a birdhouse all have a similar logical format. Consider the following recipe:

BAKED ZITI (serves 4)

Ingredients:

1 (8-ounce) package ziti
2 cups Italian tomato sauce
1 cup shredded mozzarella cheese

Actions:

1. Cook ziti according to spaghetti recipe on page 343.
2. Prepare Italian tomato sauce according to recipe on page 186.
3. Combine ziti, tomato sauce, and 1/2 cup shredded cheese in 2-quart casserole.
4. Sprinkle remainder of cheese on top of casserole.
5. Heat casserole uncovered on full power for ten minutes or until cheese is melted and sauce is bubbly.

The structure of this recipe has many parallels with a computer program. Like the recipe, a Pascal program starts with a heading. Like the recipe, a Pascal program will follow the heading with a list of its "ingredients," definitions of the objects on which it will operate.

Notice that the recipe refers to subrecipes on other pages of the cookbook: the procedures for cooking the ziti and preparing the Italian tomato sauce. The single instruction in action 2 in the main recipe stands for many definitions and actions defined in the Italian tomato sauce procedure on page 186. Many cookbooks define a set of useful procedures, functions, and subrecipes which the author invokes again and again. Procedures for stuffing and trussing a bird might appear at the beginning of the poultry section. Later in the cookbook, recipes for wild duck, turkey, and partridge can invoke these predefined procedures. The author will give gravy and sauce subrecipes and then invoke them in main recipes with shorthand phrases like "White Sauce III, page 285." Pascal offers two forms of subprogram (the *procedure* and the *function*) which are analogous to these subrecipes. They serve the same purpose: they make the main program more compact and more comprehensible.

A particularly useful kind of instruction in a recipe is one that makes the actions of the cook contingent on the condition of the item under preparation. In the ziti recipe, action 5 contains such an instruction: the cook is to heat the casserole "until cheese is melted and sauce is bubbly."

Other examples from a popular cookbook are: "simmer celery until tender"; "beat the batter until it is smooth"; "whip the egg whites until they stand in peaks." Pascal has two kinds of instructions for controlling repetitive actions. Borrowing the Pascal keywords, they are:

```
while "the batter is lumpy" do
    "beat the batter."
```

and

```
repeat
    "simmer the celery"
until "the celery is tender."
```

Assignment of Values

The above is pidgin Pascal. Here's some real Pascal:

```
repeat
    X := X / 2
until X < 1
```

As you read that, remember that every trade and profession has special jargon and symbols. Computing (and cookery!) is no exception. The special symbol := in the above statement is called the *assignment* operator. The assignment operator instructs the computer in a two-stage process:

- Evaluate the expression on the right-hand side; that is, find the value of X divided by 2.
- Assign this value to the variable whose name appears on the left-hand side.

How do you pronounce X := X / 2? Programmers usually just say "X equals X over two." Of course, that sentence doesn't make mathematical sense; unless X is zero it cannot possibly be "equal" to itself divided by 2. Another way to pronounce an assignment statement is to say "X *gets* X over 2." This emphasizes the real meaning of the Pascal statement: the value of X/2 is to be computed, and that value is to be stored in the variable named X. Pascal will always try to remind you that the assignment operator is not the same as "=" or "equals"; that's precisely why it uses := for the assignment operator. If you want to be strictly correct (and sound rather pedantic!), you might say "replace the old value of X with the old value of X divided by two."

Repeated Actions

Recall the whole statement we are talking about:

```
repeat
   X := X / 2
until X < 1
```

What does that **repeat-until** statement tell the computer to do? It says to repeatedly halve the value contained in the variable X until the value is less than one. If the initial value in X is 10, then the sequence of values assigned to X is: 5.0, 2.5, 1.25, 0.625. The final value assigned to X by this process is 0.625. With that assignment, X's value becomes less than 1, the condition of the repetition is satisfied, and the statement is complete. The computer will cease the repetition, just as the cook will stop heating the ziti when the cheese begins to bubble.

A simpler kind of repetition in a recipe is noncontingent: the action is repeated for a fixed number of times or for a given duration. Cookbook examples include "stir mixture ten times," "blend for six minutes," "rinse in clear, cold water three times." Pascal has a control structure analogous to this. Here's some pidgin Pascal translating two of the above instructions.

```
for J := 1 to 10 do
   "stir the mixture";
for J := 1 to 3 do
   "rinse in cold, clear water";
```

The words in bold letters are reserved words of the Pascal language, which are always spelled and used in the same way. Here's some real Pascal.

```
for J := 1 to 4 do
   writeln('This is a test.');
```

This Pascal instruction writes the lines

```
This is a test.
This is a test.
This is a test.
This is a test.
```

on the display screen.

Conditional Actions

In both recipes and programs, there are times when an action should be done or not done, depending on some condition. From a recipe book we have "pare apples only if the skins are tough" and "if the caramel is